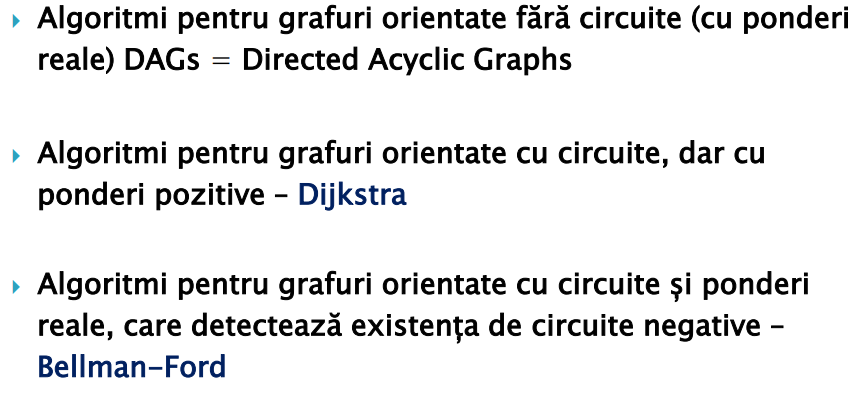
Drumuri minime in grafuri orientate ponderate (=cu costuri pe muchii = w[])

* intre doua noduri date s si v -> Dijkstra
* intre un nod sursa si toate celelalte noduri



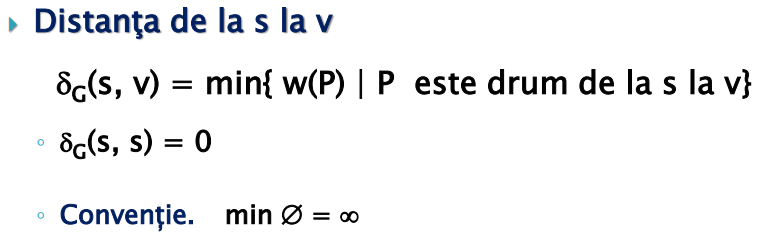
* intre oricare doua noduri -> Dijkstra / Floyd–Warshall

Obs1: daca graful nu e ponderat, pb se rez cu BFS din vf sursa

Obs2: daca la Dijkstra toate costurile sunt =, alg e echiv cu BFS

1. **Introducere**

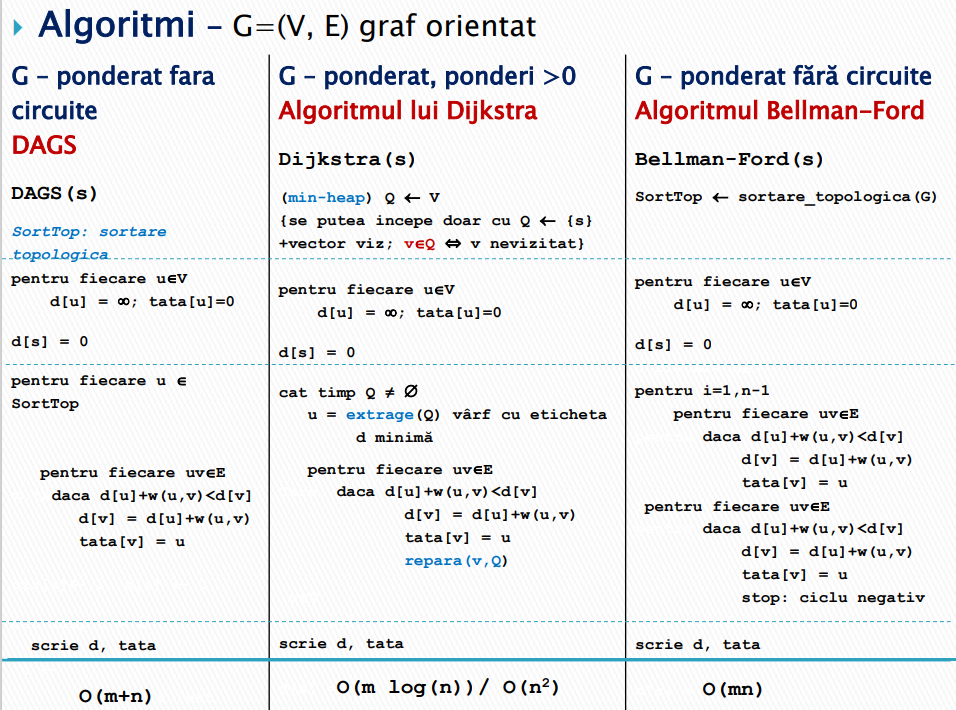
Def: Distanta de la un nod sursa la un nod v este drumul minim de la sursa la v.

* (d grecesc)

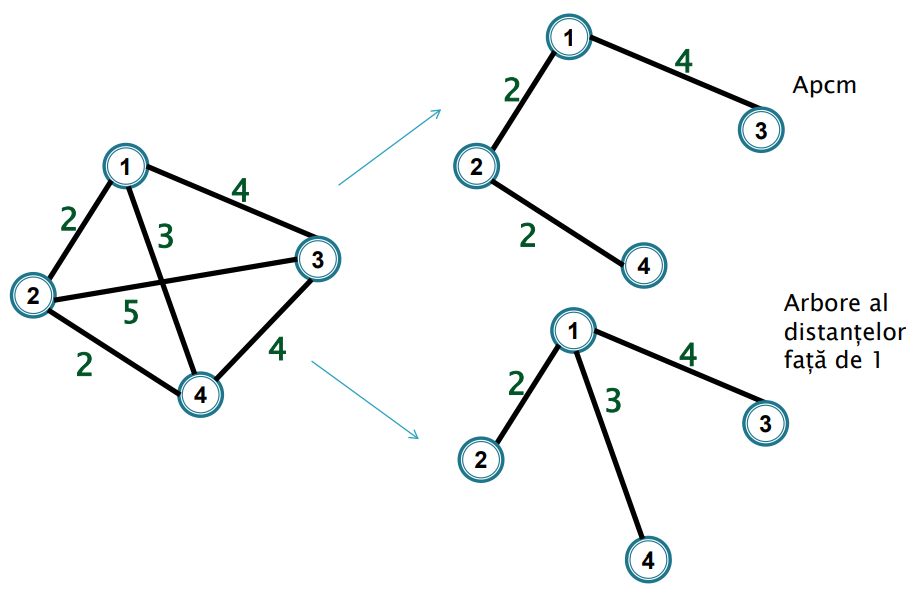
**Observatii**:

1. Drumul minim intre 2 noduri dintr-un graf fara cicluri de cost negativ = drum elementar.
2. Subdrumul unui drum minim e tot drum minim. (logic ca se schimba nodurile la care ne raport.)
3. **Drumuri minime de la un varf dat (sursa unica) la (toate) celelalte varfuri**

**=> arbore de dist**



Def: **Arbore al distantelor fata de o sursa s** = arbore cu radacina in s, format din drumurile minime de la s la celelalte varfuri accesibile din sursa (pt ca graful e orientat)

Obs: APCM nu coincide neaparat cu arborele distantelor <=> cost minim != distante minime pornind din sursa. ex: 

Abordare:

* Pentru a calcula distantele fata de s, mergem *din aproape in aproape*

=> sortam topologic varfurile pt a ne ajuta de predecesori, dar pt asta trebuie sa nu avem circuite <=> DAGs

obs: varful sursa s nu trebuie sa fie neaparat primul in sortarea top

* Dacă există circuite – *estimăm* distanţele pe parcursul algoritmului şi considerăm vârful care este estimat a fi cel mai aproape de s.

Algoritmi:

1. DAG = Directed Acyclic Graph = grafuri orientate fara circuite

-> sortare top

1. **Dijkstra** = generalizare a BFS doar cu nodurile pt. care am gasit minimul

-> din s intai pe eticheta minima (d) spre v, apoi actualizam d pt toate nodurile adiacente cu v si selectam in continuare varful cu cea mai mica val => repetam acesti pasi si pt el (ca si cum el e s acuma)

-> relaxarea unui arc (u,v) = daca putem gasi un drum mai bun de la s v, trecand prin u -> imbunatatim d[v] = d[u]+w[u,v]

-> d = distanta/costul minim de la sursa pana la un varf

w = costul unei muchii

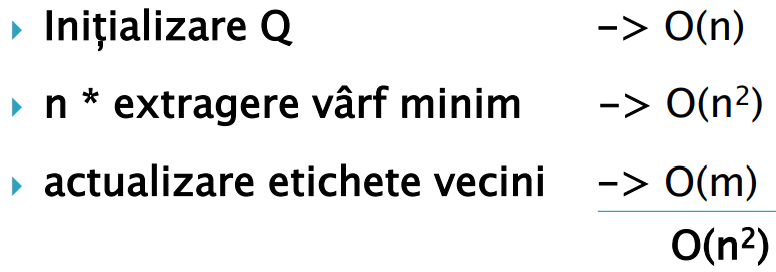
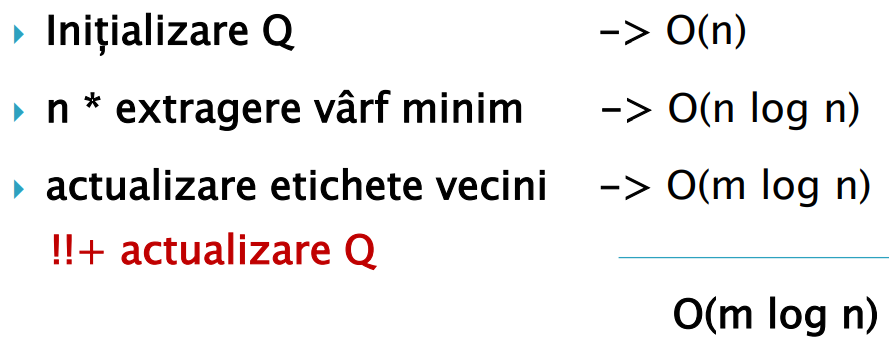
Observații: - Dacă vârful u curent are eticheta d[u] = ∞, algoritmul se poate opri

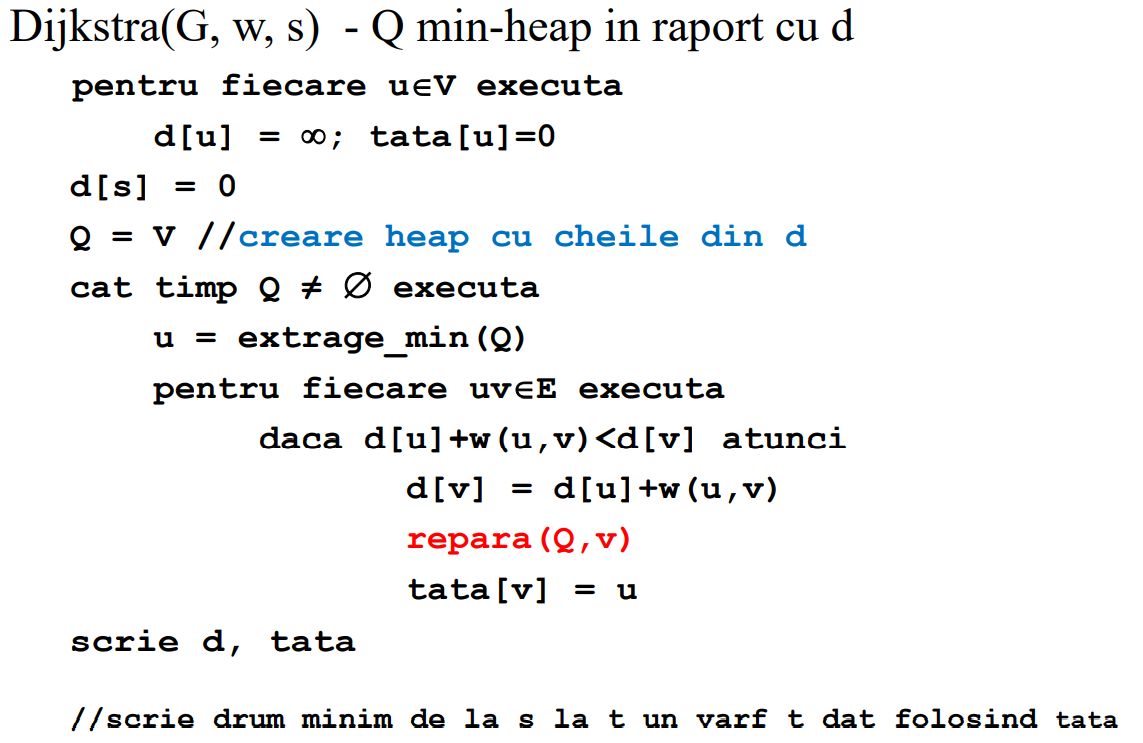
- Vectorul tata memorează arborele distanțelor față de s (vf inaccesibile din s rămân cu tata 0)

Complexitate:

- Q = vector de 0 si 1 => O(n2)

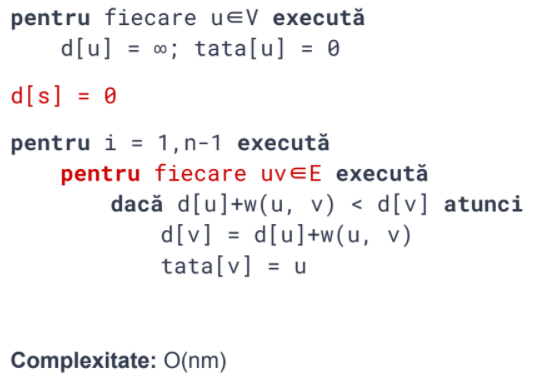
- Q = min-heap => O(m log n)

 => O(m log n)

1. **Bellman – Ford**

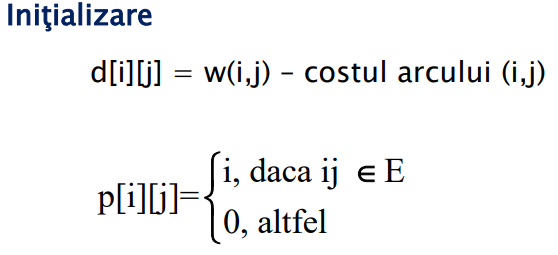
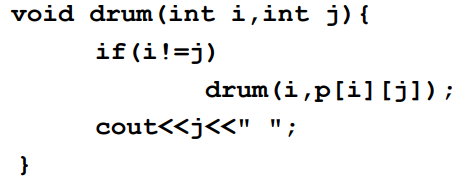
Dumitran: La un pas, relaxăm toate arcele (obs: nu relaxăm arcele dintr-un vârf selectat u, ci din toate vârfurile) => n-1 etape => după k etape, d[u] ≤ costul minim al unui drum de la s la u cu cel mult k arce (progr dinamică) ⇒ după k etape, sunt corect calculate etichetele d[u] pentru acele vârfuri u pentru care există un s-u drum minim cu cel mult k arce

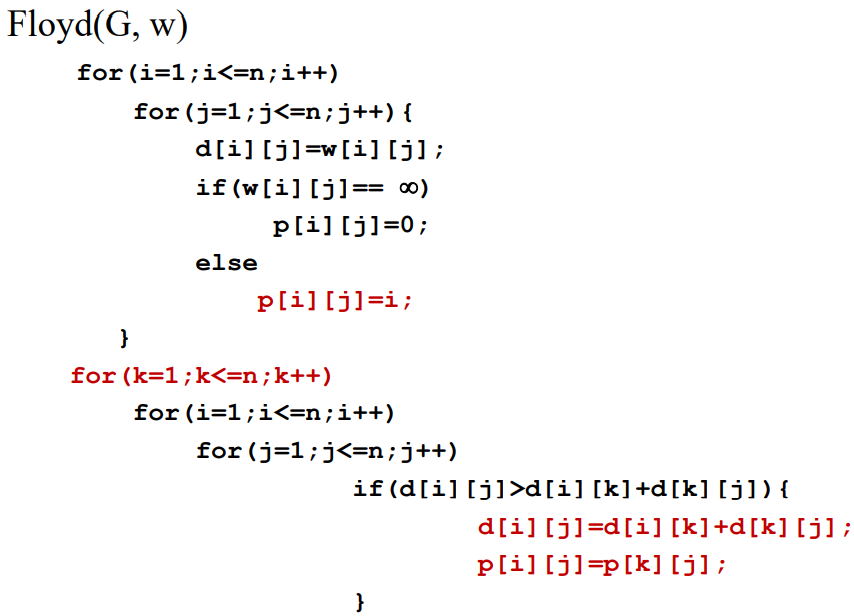


1. Drumuri minime intre 2 varfuri date

* Dijkstra cu următoarea modificare: dacă vârful u ales este chiar t, algoritmul se oprește; + drumul de la s la t se afișează folosind vectorul tata

1. Drumuri minime intre toate perechile de varfuri
2. Dijkstra -> ponderi pozitive => n\*O(mlogn)
3. **Floyd – Warshall** -> ponderile pot fi și negative dar NU există circuite cu cost negativ în graf

 Afisarea:

 **-> O (n3)**